1266 Hewlett-Packard Journal
36(1985)Febr.,No.2,Amstelveen,Netherlands

Disc Caching in the System Processing Units of the HP 3000 Family of Computers

Disc caching uses the excess main memory and processor capacity of the high-end HP 3000s to eliminate a large portion of the disc access delays encountered in an uncached system.

by John R. Busch and Alan J. Kondoff

THE PRICE/PERFORMANCE RATIO of processors and the cost per bit of semiconductor memory have been falling rapidly in recent years, while moving-head discs have been getting denser but not much faster. The resulting increase in the ratio of access times between secondary and primary storage, coupled with a reduction in the required number of secondary storage servers, has caused low processor utilization in many systems. Proposals to address this problem have focused on:

- Reducing the basic disc access time through the use of higher-speed actuators and rotation rates or through multiple actuators and heads.
- Reducing the effective disc access time by front-ending one or more discs with a semiconductor disc cache providing multiple track buffering or by inserting an intermediate level in the storage hierarchy between the discs and main memory using CCD (charge-coupled device), magnetic bubble, or semiconductor RAM technology.
- Reducing the frequency of disc access through local and global file buffering in the main memory.

The research and development effort reported here examined alternative approaches to exploit current trends in processor and memory technology to realize significant improvements in system price/performance ratio, and applied the results to the HP 3000 family of computers. The solution we selected for the HP 3000 family was to apply the excess main memory and processor capacity of the newer systems to prefetch and cache disc regions and deliver data when requested at main memory speed rather than disc access speeds.[1] The approach eliminates not only a significant portion of the traffic between primary and secondary storage, but also the bulk of process delays waiting for the traffic that remains. Through the integrated management of data base, file, and transient object space, it matches data management requirements and the architecture of the HP 3000 family with the current trends in processor and memory technologies.

In this report, we analyze the alternatives for exploiting processor and memory technology trends with respect to cost, performance, and reliability. We discuss disc caching design alternatives including fetch, replacement, and write handling policies. We present an overview of the tools

Glossary

Read hit rate. The probability that the disc file information to be read will be found in a higher-speed memory and will not have to be obtained from a slower one.

Read miss. Failure to find the desired data in a higher-speed memory.

Nowait. A type of input/output in which a process requests an I/O operation and then proceeds with other processing while the I/O operation completes.

Write wait probability. The probability that a process will have to wait for a disc write operation to complete after it has initiated a nowait write.

File mapping. A type of disc caching in which disc files are addressed as virtual memory locations, with the processor hardware and memory manager performing physical location and file caching functions.

Disc domain. A contiguous section of disc storage containing file data.

File extent. A sequence of file records stored in a contiguous disc region.

Posting. Updating a record or file in its permanent home in nonvolatile disc storage.

MIPS. Million instructions per second.

LRU. Least recently used. A criterion for removing cached disc domains from cache storage to make room for currently needed domains.

Shadow paging. A transaction recovery mechanism in which file changes of uncommitted transactions are saved in temporary disc locations until transaction commit time, when the temporary changes are made permanent.

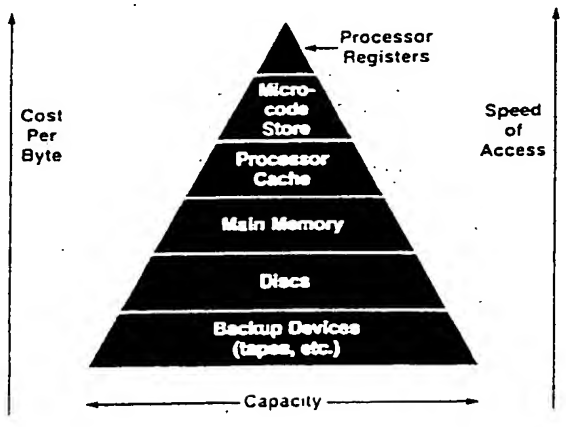

Fig. 1. *Standard computer system storage hierarchy.*

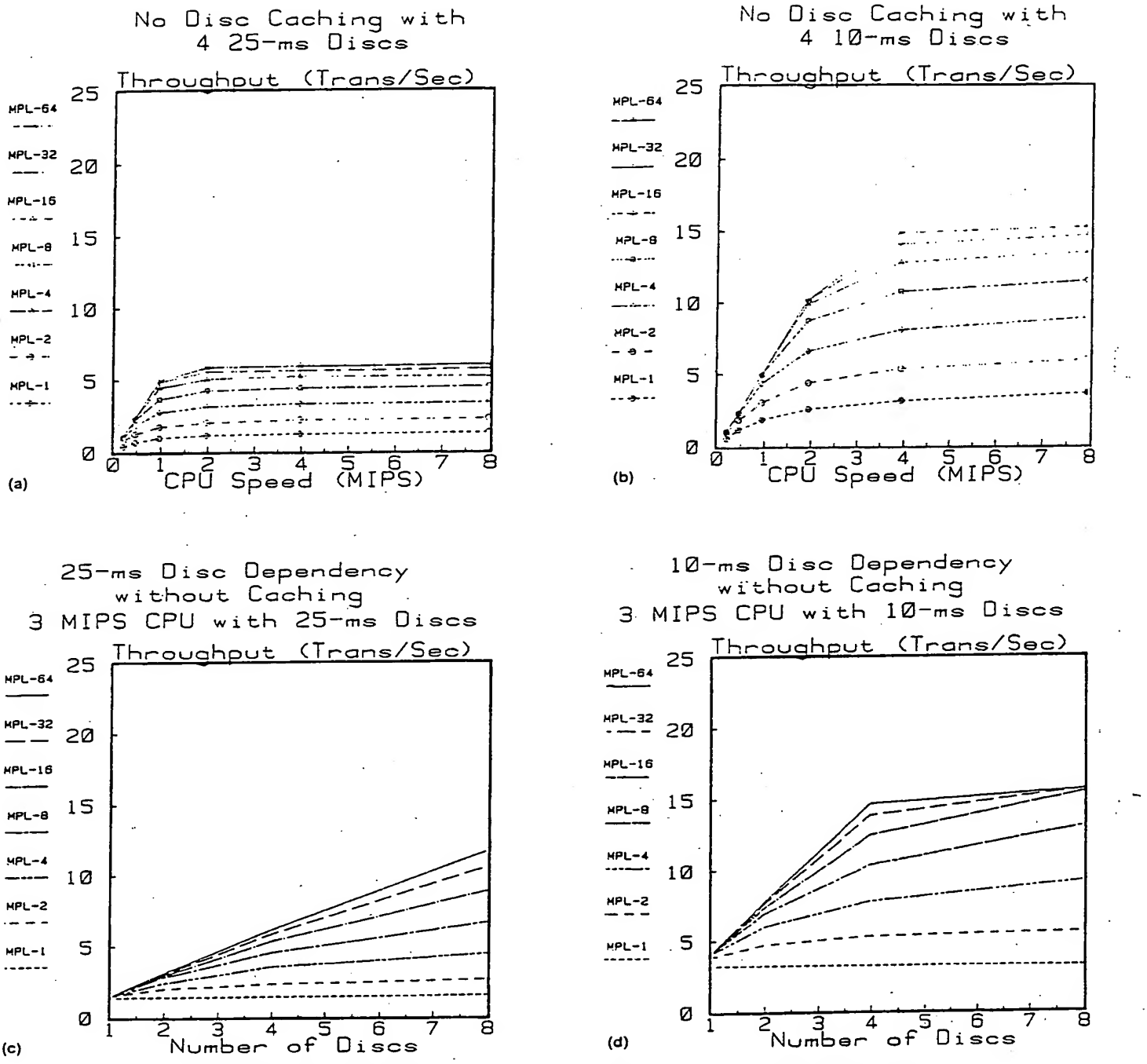

Fig. 2. *Performance of systems using conventional memory hierarchies without disc caching. Transaction throughput is shown as a function of effective multiprogramming level (MPL), the number of processes that can be concurrently executed. The top graphs show throughput versus CPU speed for a fixed number of discs. The bottom graphs show throughput versus number of discs for a fixed CPU speed.*

developed for the performance analysis of the alternatives, and we present measurements of the system and subsystem behavior and performance of our implementation for the HP 3000 family.

Disc caching is standard on HP 3000 Series 42, 48, and 68 systems and on HP 3000 High-Performance Series 39 systems. It is supported by the MPE-V/P and MPE-V/P Delta 1 operating systems, and by all releases of the MPE-V/E

(continued on page 24)

Disc Cache Performance Tools

To evaluate the performance of alternative system architectures and management policies, performance measurement and analysis tools were constructed. Instrumentation was defined, a statistics updating and reporting subsystem was designed and constructed, and display tools were specified and built. Workload generators were collected, and workload characterizers were constructed. A simulation was built to analyze secondary storage caching, and a system model was constructed for analysis of the impact of disc caching across a broad range of configurations and workloads. Fig. 1 shows the relationship of the various performance measurement and analysis tools that were constructed for this research.

The measurement subsystem supports low-overhead, event-driven statistics gathering. Statistics including resource queue length and service time histograms, service class compositions, transition distributions, and resource-specific event frequencies are supported by the processor, main memory, disc, and semaphore managers. A statistics display tool allows the measurer to specify the range of statistics to be measured and the desired duration of measurement.

To analyze the disc subsystem in detail, a trace-driven disc workload characterizer and disc cache simulator were specified and built. These tools provide insight into the workload's disc traffic and the impact of various forms of disc caching. These tools consume disc access trace files as input. A trace record contains the time, accessor class, access transfer count, access function, and access location for each disc access initiated by the system being traced.

The disc workload characterizer produces a profile of disc I/O workload characteristics. This includes a breakdown of the workload into its access type components (data base, directory, sequential access, etc.), interreference times, distributions of transfer sizes, active extent sizes, and extent LRU (least recently used) stack reference depths.

The disc cache simulation allows refined control over cache management policies (rounding, extent adherence, write handling, fetch sizes for each access type, and flush control). Any subset of the accessors can be cached, so that the localities of the access methods can be investigated in isolation. The simulator decomposes the disc references into modes and access functions and gives cache hit information for each access type. In addition to the cache performance information for the access types, the disc cache simulator also gives cache behavior information, including distributions of cache entry counts, cache reference LRU stack depths, and cache interreplacement times.

The hit rates obtained from the simulations are used as the processor-to-disc and processor-to-cache transition probabilities in the system model. Since the hit rates are obtained through simulation, main memory size and contention do not have to be captured explicitly by the system model. This significantly reduces the complexity of the system model.

A custom analytic system model was required so that a disc cache service station could be explicitly included. This allows the user to specify the system configuration and workload characteristics, and produces global performance estimates as well as resource demand and utilization statistics. The system model explicitly captures the effects of alternative secondary storage caching mechanisms, including external caching of discs through an intermediate storage level and internal caching of discs through file mapping or explicit caching in primary memory.

The system model queries for workload and configuration information. These inputs are obtained from knowledge of the installation, from the disc cache simulation, and from the statistics collection and reduction tools. Inputs are specified for processor speed, disc configuration and speed, channel speed, processor and disc service demand, disc cache overhead, and disc cache

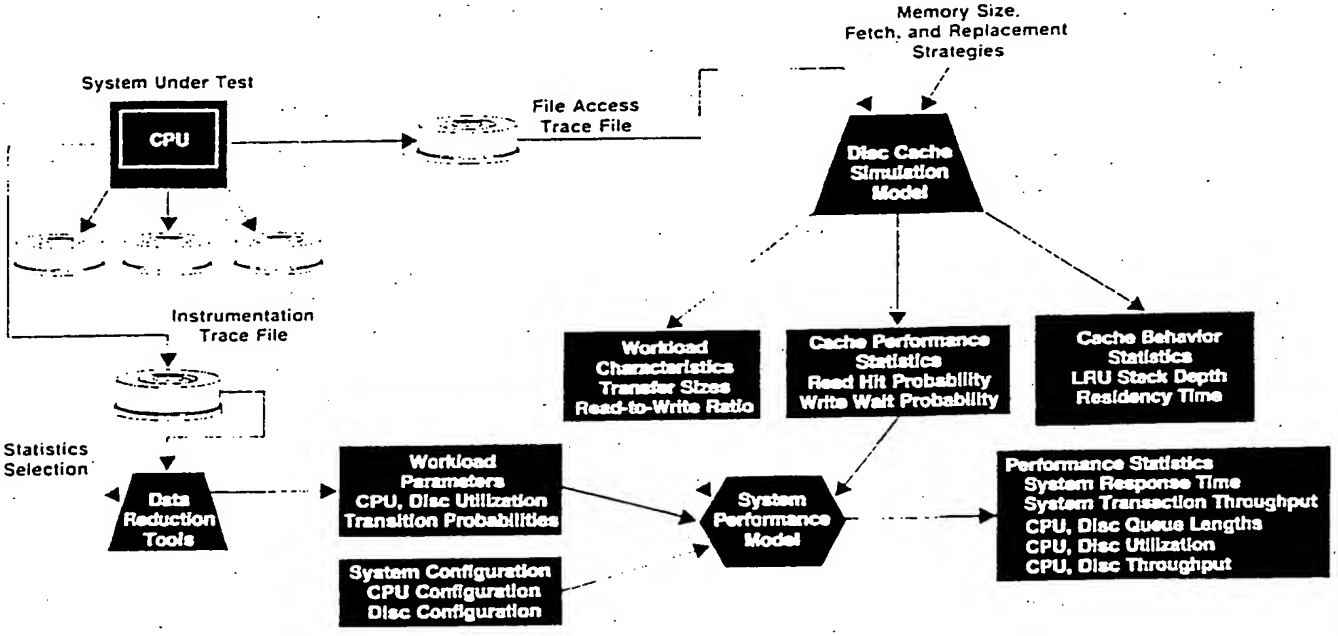

Fig. 1. *Performance measurement and analysis tools developed to evaluate alternative system architectures and management policies*

read hit rate, fetch size, and post handling method.

Given the input system configuration and workload characterization, the system model program constructs a closed queueing model. The total workload is aggregated, and the network is decomposed into a simple central server model. For internal caching, the disc cache service station is absorbed into the processor station. The resulting central server model is solved through a standard iterative procedure[1] which iterates on the effective multiprogramming level.

Performance and behavioral characteristics are calculated for each server and for the system as a whole. For each iteration on the mean multiprogramming level, the utilization, mean queue length, throughput, and response time of each server are calculated, as are the overall system response time and throughput.

Serialization caused by contention for access control semaphores (e.g., data base locks) can significantly limit the effective multiprogramming level. The impact of serialization is not explicitly captured by the model. Its effects can be approximated by reducing the effective multiprogramming level.

Reference

1. S.S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, New York, 1983.

(continued from page 22)

operating system. It includes software, additional memory, and in some cases, additional firmware.

Technology Trends

Fig. 1 (page 21) shows a conventional computer system storage hierarchy. Storage hierachies provide cost-effective system organizations for computer systems. Each successive level of a storage hierarchy uses lower-cost, but slower, memory components. By retaining frequently accessed code and data in the higher-speed memories, the system can operate at speeds close to the access times of the fastest memories while most of the storage capacity is in lower-cost, lower-speed memories. The price and performance of a computer system are often dominated by the organization and management of its storage hierarchy.

Achievable system performance is a direct function of processor speed and utilization. Processor utilization, and so the effective exploitation of high-speed processors, is limited primarily by the waiting time caused by misses at various levels of the storage hierarchy (i.e., the desired data is not found in a higher-speed memory and must be obtained from a slower one). Thus, for optimal system price/performance ratio, the processor speed and the capacity, speed, and management of the levels of the storage hierarchy must be matched.

Traditional solutions for a low hit rate at a certain level of the storage hierarchy include improving the management policies of the levels, increasing the capacity of the level incurring the low hit rate, speeding up the access time of the next lower level of the hierarchy, and introducing a new level into the storage hierarchy. Cost and technology determine which alternative or combination of alternatives is appropriate.

Processor speeds are rapidly increasing, and costs are dropping. The order-of-magnitude advances in processor speeds in recent years have not been matched by proportional advances in disc access speeds. There is an access time gap of five orders of magnitude, and widening, between a main memory reference and a disc reference in most current computer system storage hierarchies. By comparison, the access time gap between a processor cache reference and a processor cache miss resulting in a cache block fetch from main memory is normally less than an order of magnitude.

Disc densities are rapidly increasing, and large-capacity discs offer a fourfold price advantage over small-capacity discs. This trend in disc technology exacerbates the hierarchy imbalance. Since the disc subsystem often represents over 50% of the system cost, the trade-off towards a small number of high-capacity discs is attractive. However, the demand for a disc grows with its capacity. Thus, realizing the cost advantages of a few large-capacity discs over several small-capacity discs reduces the potential for parallel service. This increases the mean disc queue lengths, and thereby the expected values for disc service response time.

Replacing discs with secondary storage devices employing CCD, magnetic bubble, or semiconductor RAM technology has shown limited cost-effectiveness. Bubble and CCD memories have not been able to keep pace with the density improvements and resulting drop in cost per byte of semiconductor RAM, while the cost per megabyte of semiconductor RAM is still two orders of magnitude greater than that of discs. Leading-edge technologies[2,3] offer the potential for high-density discs with mean access times approaching 10 ms. Thus, magnetic-media secondary stor-

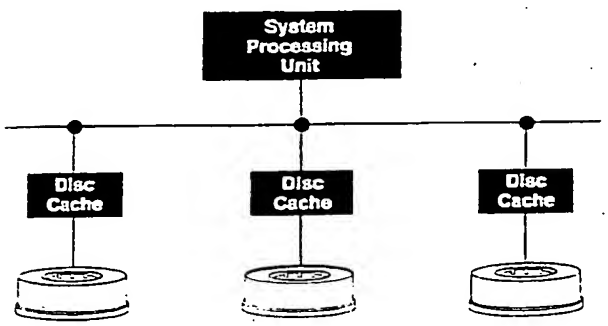

Fig. 3. *External disc caching using a local buffer for each disc.*

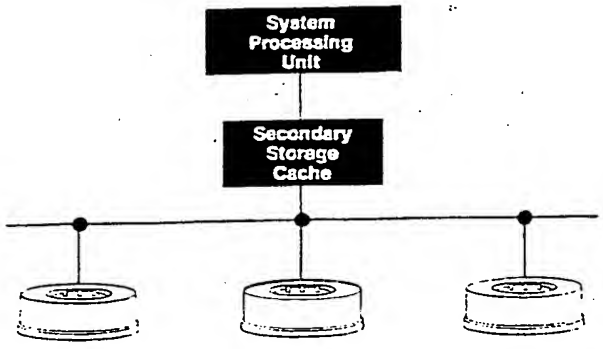

Fig. 4. *External caching at an intermediate storage level.*

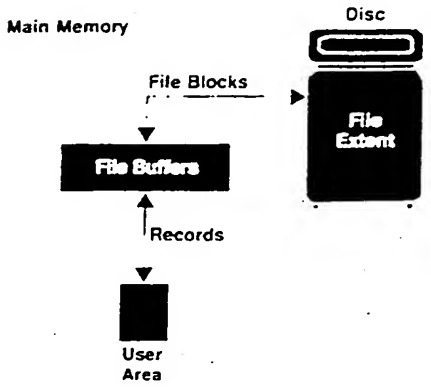

Fig. 5. *Many systems provide limited disc caching in main memory on a subsystem, file, or application basis.*

The MPE-IV Kernel

The kernel of an operating system provides higher-level system software including applications, data bases, and file systems with a virtual interface to the hardware and primitive system resources. The details of system configuration, of hardware interfacing, and of controlling access to system resources are hidden by the kernel.

The kernel multiplexes the system resources among the competing processes through the application of service policies. The overall performance of a general-purpose computer system is to a large extent determined by the effectiveness of these policies.

As the HP 3000 began to expand into a family of computers, a new kernel was required that would support its evolution. A research effort was undertaken, resulting in the MPE-IV kernel. The research approach and the design and performance of the resulting kernel are described in reference 1. Our implementation of disc caching for the HP 3000 family naturally extends the algorithms, measurement tools, and facilities of this kernel.

The kernel structure is based on decomposing the system into resource management modules that can be easily replaced, while enforcing within a module strong separation of policy from mechanism so that new algorithms can be implemented in a leveraged manner. Performance was sought through good algorithms, not instruction-level optimization.

Algorithms for main memory management, processor management, disc access and space management, interprocess communication, and semaphore management were developed that cooperate to optimize global system performance objectives rather than local performance objectives.

To achieve this cooperation, global priority assignments are made to processes based on external policy specifications. These process priority assignments are then reflected in the service requests for the various system resources so that local management decisions contribute to the global objectives. The resource management algorithms of the MPE-IV kernel are discussed in detail in reference 2.

References

1. J.R. Busch, "The MPE IV Kernel: History, Structure and Strategies," *Proceedings of the HP 3000 International User's Group Conference*, Orlando, April 27, 1982, reprinted in *Journal of the HP 3000 International User's Group*, Vol. 5, no. 3 4, July/December 1982.

2. J.R. Busch, *Integrated Resource Management Algorithms for Computer Systems*, Ph.D. Dissertation, University of California, Los Angeles, 1984. University Microfilms International, Ann Arbor, Michigan, no. 8420156.

age devices and semiconductor primary memories can be expected to remain dominant in computer storage hierarchies in the years to come.

The results of the direct application of evolving technology are shown in Fig. 2 (page 22). These graphs use the analytic system model described in the box on page 23. The graphs show families of curves for increasing effective multiprogramming level (the number of processes in memory demanding service that can be concurrently executed). The top graphs show transaction throughput as a function of processor speed for a fixed number (4) of 25-ms and 10-ms discs. The bottom graphs show transaction throughput as a function of the number of 25-ms and 10-ms discs at a fixed processor speed of 3 MIPS.

A fixed channel bandwidth of 2M bytes/s is assumed for all the runs. Other assumptions were 10,000 instructions per disc visit, five reads per write access, 1K-byte mean transfer size, and five disc accesses per transaction. These are somewhat characteristic of the referencing patterns of the HP 3000 family, although observed variations are wide. Changes in these parameters shift the curves, but the general characteristics are the same.

We see that with conventional storage hierarchy management with this workload and with four 25-ms discs, effective processor utilization extends only to 1 MIPS, and beyond 1 MIPS, performance is linear with respect to the number of discs. With faster discs (four 10-ms discs), effective processor utilization extends through 4 MIPS, with a sharp dependency on the number of 10-ms discs for the higher multiprogramming levels. High effective multiprogramming levels are required throughout to exploit processor capacity.

This indicates that for effective utilization of higher-speed processors using conventional storage hierarchies and management techniques and high effective multiprogramming levels, ever faster discs in greater numbers are required. As discussed above, the technology trends in secondary storage devices do not support these requirements.

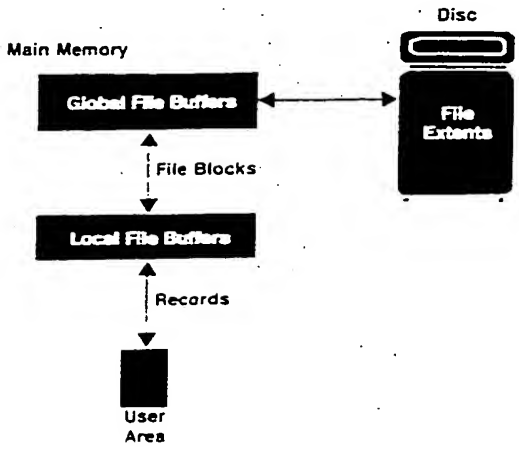

Fig. 6. *Some systems provide centralized buffering schemes, setting aside a portion of main memory for global file buffers.*

Alternative Balancing Approaches

System throughput is directly proportional to effective multiprogramming level, and inversely proportional to processor and disc response times and disc visit frequencies. Consequently, efforts to overcome the limitations in exploiting the trends in processor and memory technologies have focused on:

- Reducing the effective secondary storage access time through external disc caching techniques
- Reducing the number of secondary storage visits per transaction through internal disc caching in the primary memory
- Sustaining large effective multiprogramming levels through improved concurrency management schemes.

In the next two sections, methods of reducing effective disc access time and visit frequencies are examined. Modern methods for achieving high concurrency with data access in transactional systems are addressed in references 4 through 8.

External Caching Techniques

Caching techniques have been applied to the disc subsystem to reduce the effective disc access time to secondary storage. Caching has been implemented in discs, in controllers, and as stand-alone system components interfacing one or more secondary storage devices to the main memory.

The use of a local cache per disc is depicted in Fig. 3. Smith[9] discusses buffering discs using bubble, CCD, or electron beam memories. He concludes that three buffers, each a cylinder in size, would produce a hit ratio on the order of 96%, with LRU (least recently used) working well as a replacement policy. IBM has announced an intelligent cached disc featuring a 384K-byte microprocessor-driven controller that optimizes seeks and caches recently referenced data.[10] Krastins[11] discusses a cache consisting of 1M to 2M bytes of RAM that is integrated with the disc controller. The cache buffers full physical tracks. He reports a hit rate of 85%, a mean access time of 8 to 12 ms, and hit processing time less than 1 ms.

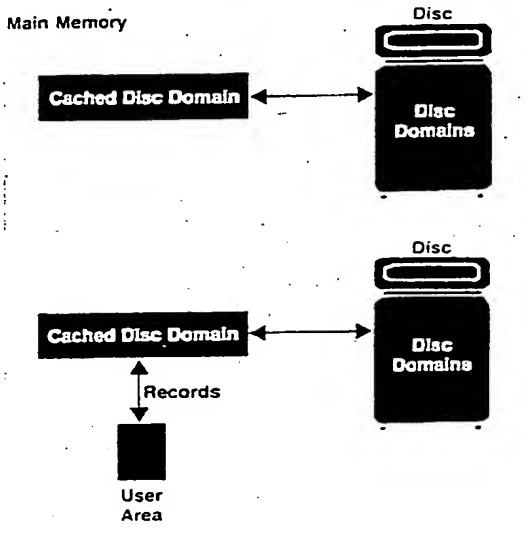

Fig. 7. *Explicit global disc caching in main memory.*

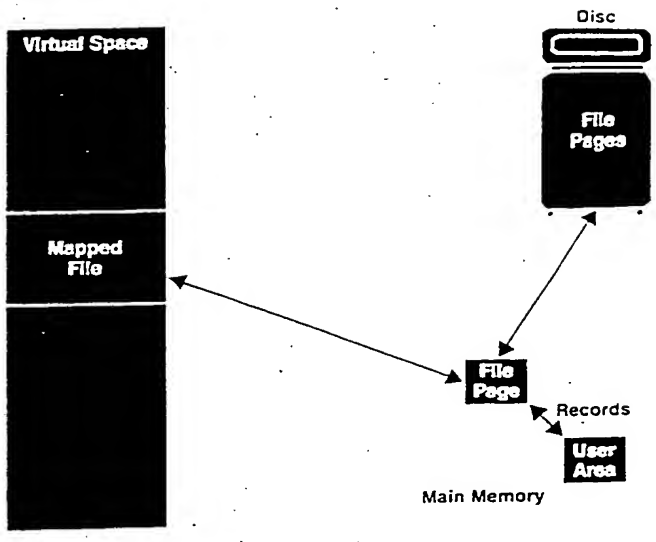

Fig. 8. *Internal caching by means of file mapping.*

The use of a cache front-ending the disc subsystem is depicted in Fig. 4. This can be viewed as inserting a new level into the storage hierarchy.

The use and organization of a CCD memory level that could serve to close the gap between semiconductor and disc access times are discussed in references 12 and 13, and the use of magnetic bubble memory systems for this purpose is discussed in references 14 and 15. But since bubble and CCD memories have not been able to keep pace with the density improvements and the drop in the cost per byte of semiconductor RAM, they have not qualified in gap-filling efforts. Rather, the technology of choice for external disc caches has been semiconductor random access memory.

Hugelshofer and Schultz[16] describe such a semiconductor disc cache marketed by Computer Automation, Inc. It consists of 2M bytes of RAM placed between the processing system and up to four moving-head disc drives. They quote access times of 4 ms on a hit, with up to an 85% hit ratio. It is packaged with four RAM cards and an LSI-2 I/O processor.

The use of external caching, either locally or through an intermediate storage level, reduces the effective secondary storage access time on read hits, and potentially decreases the access time on writes, provided immediate physical update of the disc media is not required on a write access before signaling transfer completion to the processor.

Using Large Primary Memories

With the improvements in memory densities and access times, very large main memories can be cost-effective provided they can be exploited to reduce the traffic between main and secondary storage. Techniques to exploit main memory for this purpose include auxiliary local buffering in applications and subsystems and global disc caching through explicit caching or file mapping.

Systems have conventionally provided limited caching of the discs in main memory through explicit buffering on

a subsystem, file, or application basis as shown in Fig. 5.

Centralized buffering schemes are employed in the Berkeley 4.2 UNIX file system for the Digital Equipment Corporation VAX computer family[17] and in a product for the IBM Personal Computer.[18] As shown in Fig. 6, a fixed portion of main memory is set aside for global file buffers. When a file block read from disc is requested, the global buffers are checked first. If the requested block is present in a global file buffer, the read is satisfied with a move from the buffer to the user's address space. Otherwise, a global buffer is freed, a disc read of the block is initiated into the selected global buffer, and when the read completes, the data is moved into the user's space. File block writes are performed through global buffers as well.

In global disc caching, main memory partitions disappear and cached disc regions containing file data are centrally managed with the pieces of transient objects by the main memory manager. In this approach to disc caching in main memory, pieces of the disc are mapped as data objects, and placed and replaced by the normal memory management algorithms like those used for code, stacks, etc. This approach is shown in Fig. 7.

Global disc caching in main memory can be either explicit beneath the disc access interface or implicit

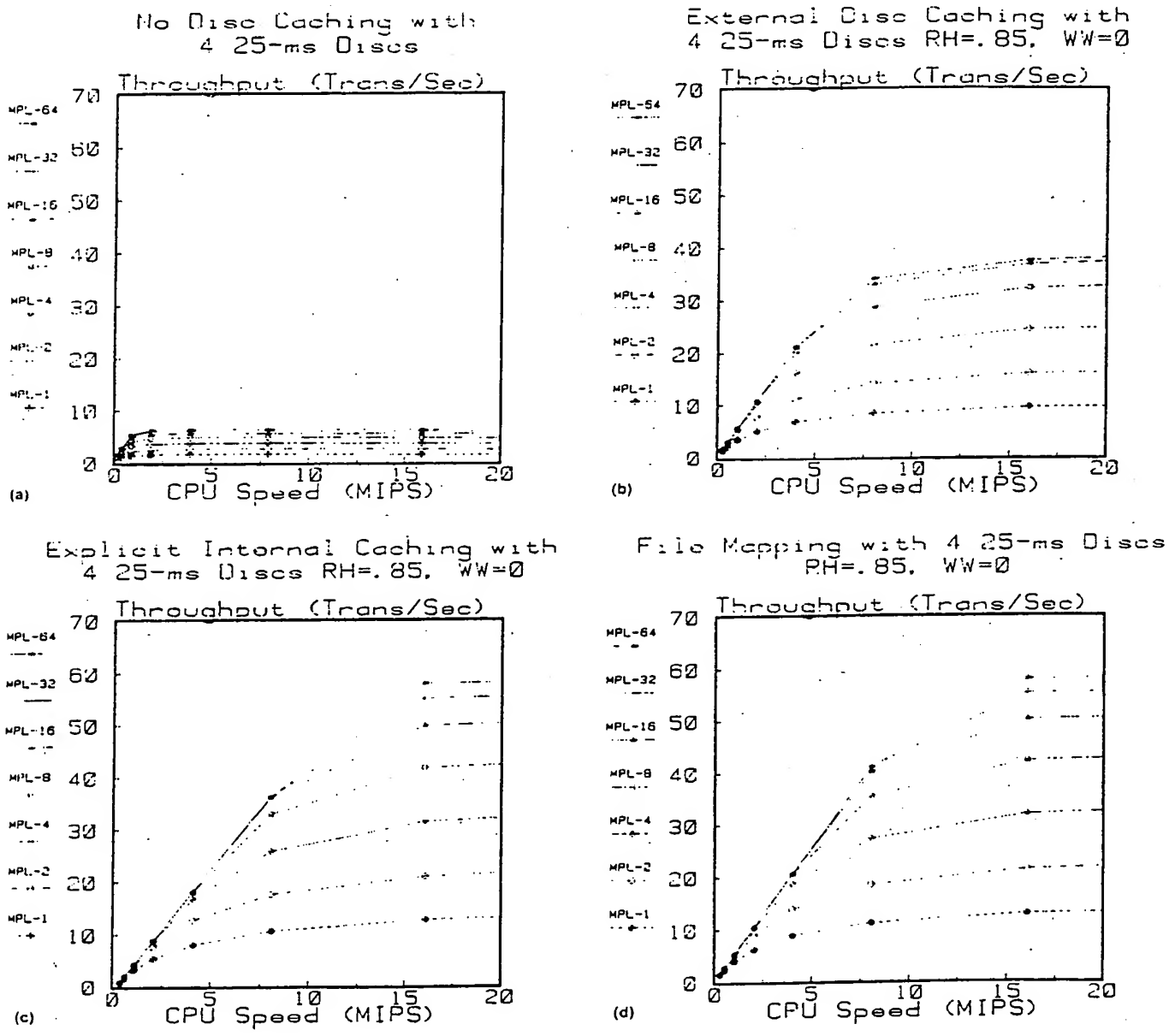

Fig. 9. *Performance of various disc caching alternatives compared to conventional storage management. Assumptions are four 25-ms discs, read hit rates of 85%, and write wait probabilities of zero.*

through the use of file mapping.

With explicit disc caching in main memory, the access methods continue to address the discs, but a software translation mechanism locates the required cached disc domain in main memory and moves the data between the cached region of the disc and the data area of the access method. This approach was implemented in the breadboard system for experimentation purposes, as described later in this report.

With architectures supporting large virtual address spaces, pieces of files or discs or entire files or discs can be mapped directly into the address space. The location of a piece of a file or disc in main memory is then handled by the virtual-to-physical address translation hardware, and the normal memory management mechanisms handle fetching and replacing of pages of files or discs. This approach to secondary storage caching is depicted in Fig. 8. This approach was first employed in Multics,[19] and more recently as described in reference 20.

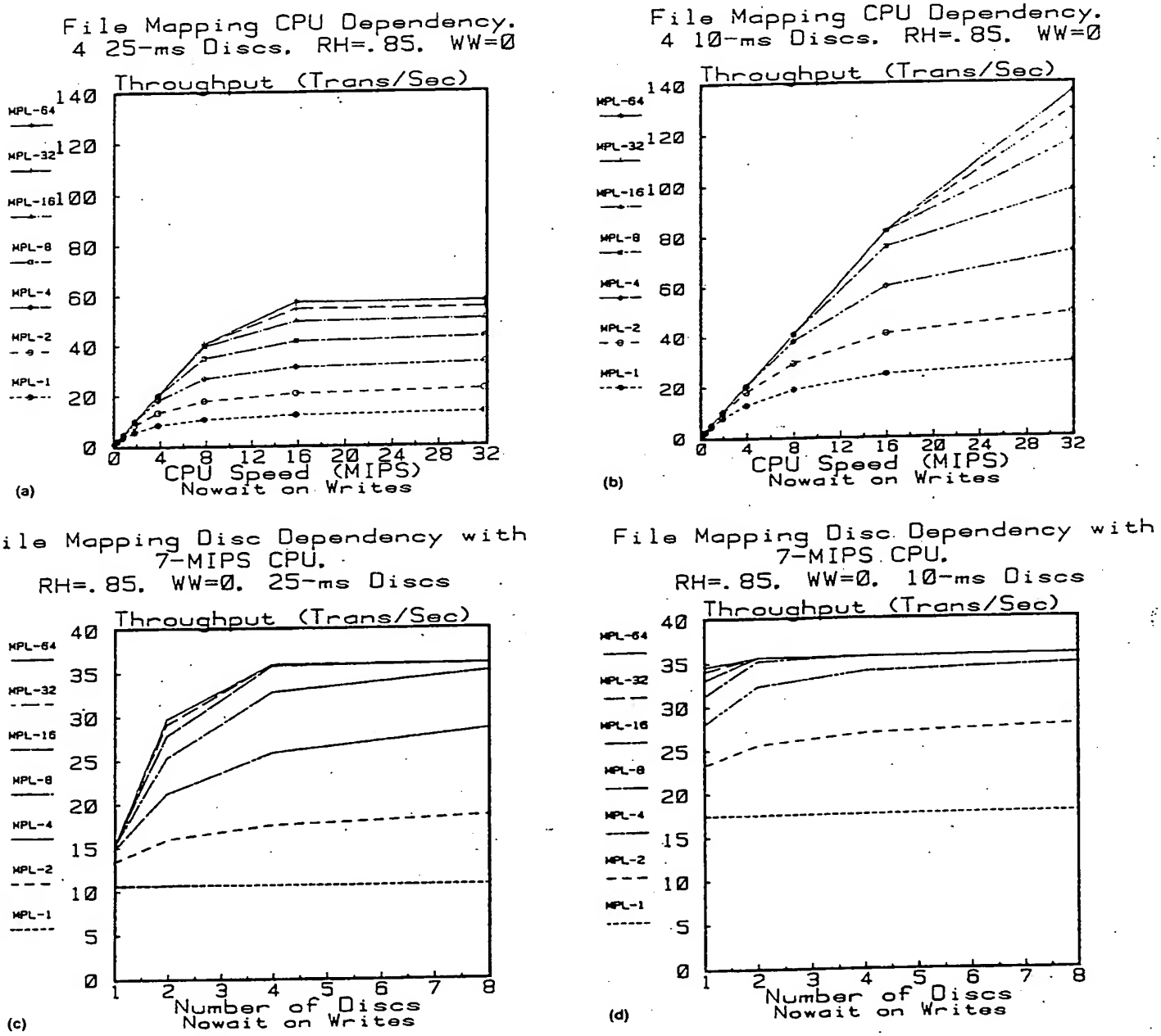

Fig. 10. *Dependency on disc access time and the number of discs is dramatically reduced with disc caching, as shown here for file mapping.*

Requirements of Transaction Management

Modern transactional data base systems guarantee that the data base is left in a consistent state in the event of a transaction abort or a system/disc failure. Solutions to the problem of balancing the storage hierarchy must preserve the consistency and recovery principles of the transactional system.

The standard mechanism used to achieve transaction recovery is the use of a write-ahead log.[21] In write-ahead logging, before the data in the data base is modified, copies

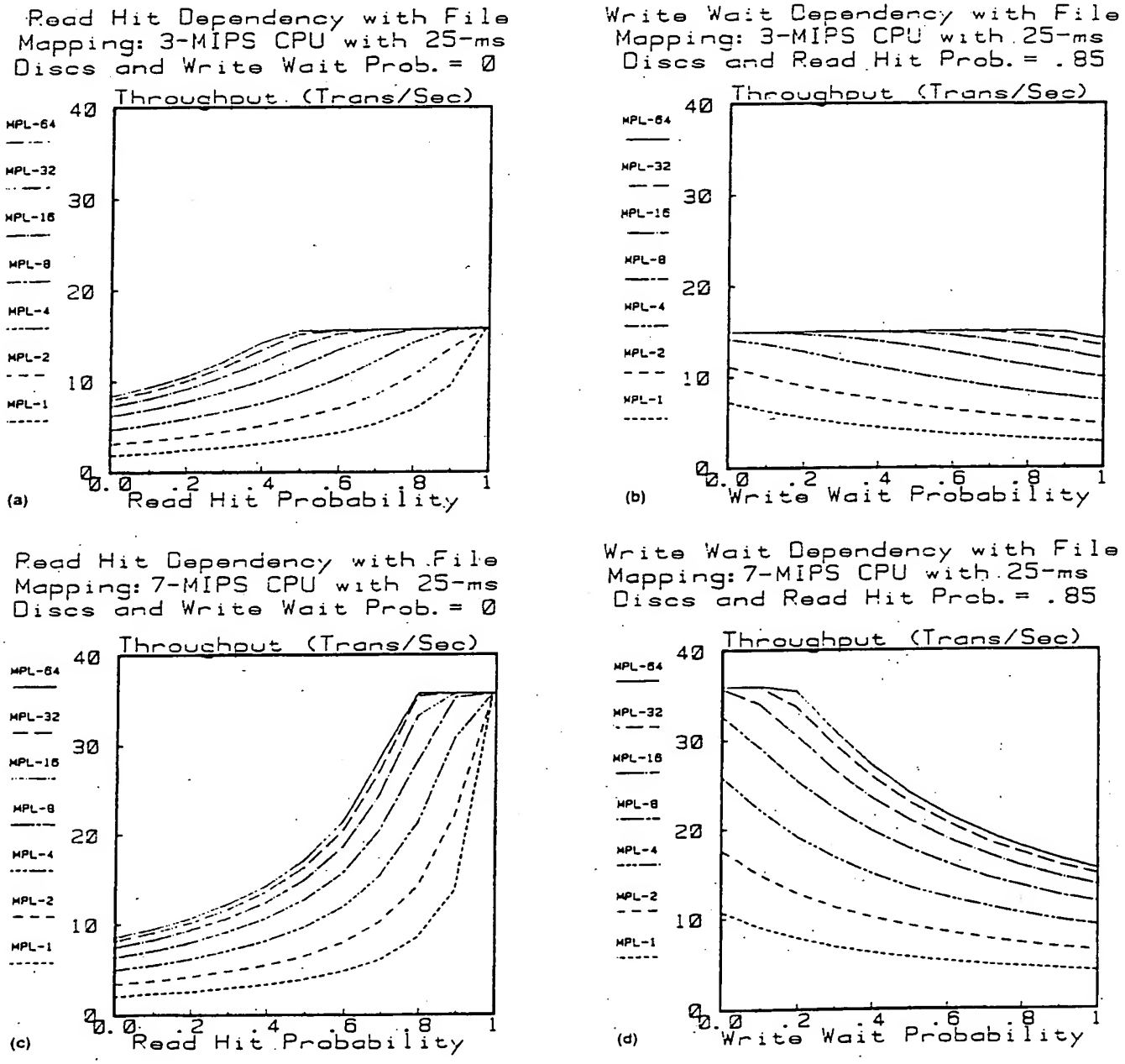

Fig. 11. *Sensitivity of file mapping to read hit rates and write wait probabilities increases with processor speed. A system using the internal disc caching techniques needs to exploit the known methods of achieving high read hit rates and low write wait probabilities.*

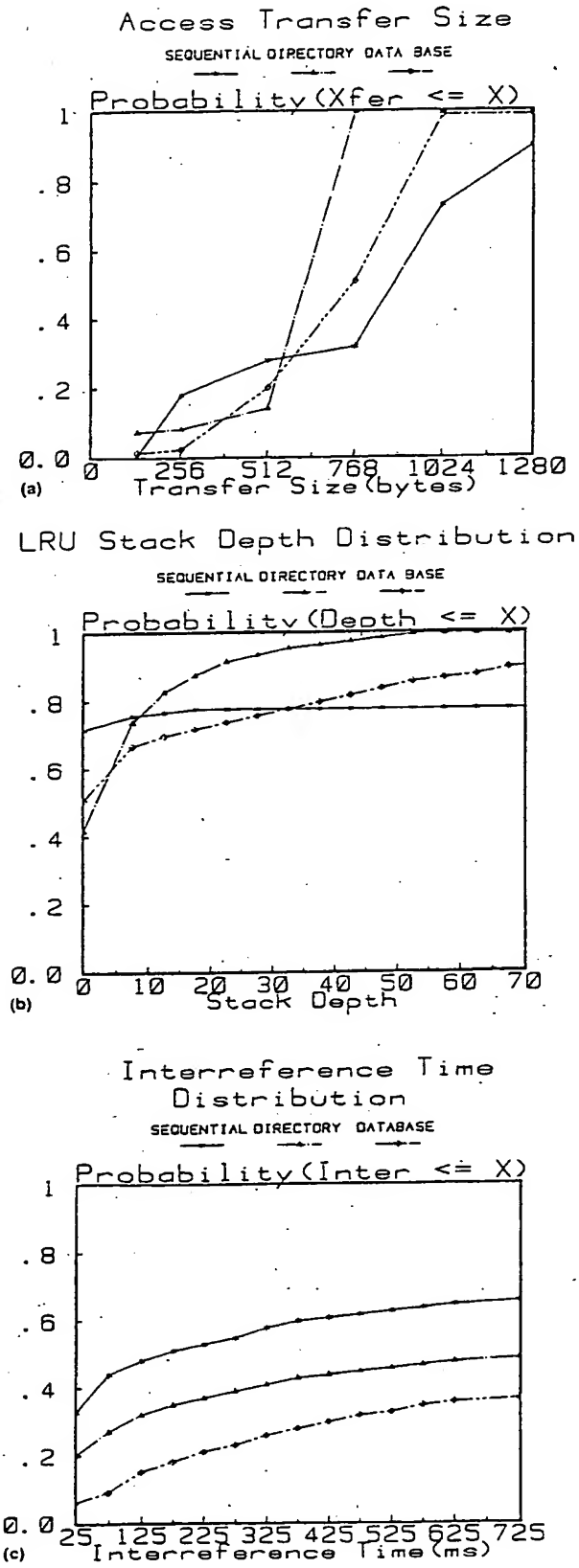

Fig. 12. *Workload characteristics of the HP 3000 family of computers without disc caching.*

of the old and new images of the data are written to a log file, along with identification information. If the transaction aborts, the old images of data modified by the transaction are read from the log and restored into the data base, thereby "undoing" the actions of the transaction. If the system crashes, a utility goes through the log file and undoes any actions of uncommitted transactions. If a disc fails, the data base is restored to its last backed-up state and the actions of committed transactions are redone using the log file.

For transaction recovery through write-ahead logging to work, the before and after images of the data along with identifying information must be posted to the disc before the data base is updated with the new values. Most data base systems issue the log write, then wait for the physical post to complete before issuing the data base write.

Comparative Analysis of Alternatives

In the following sections we evaluate the alternatives discussed above based on performance, cost, and reliability. We identify complementary approaches, and identify workload and configuration sensitivities affecting the suitability of alternatives.

Evaluation with Respect to Performance

Evaluating with respect to performance, we saw in the preceding section that effective processor capacity utilization drops rapidly with increasing processor speed if conventional storage management is employed. Faster discs and more discs provide benefit in such systems, but the benefits are not commensurate with those obtained through secondary storage caching. This is demonstrated in the

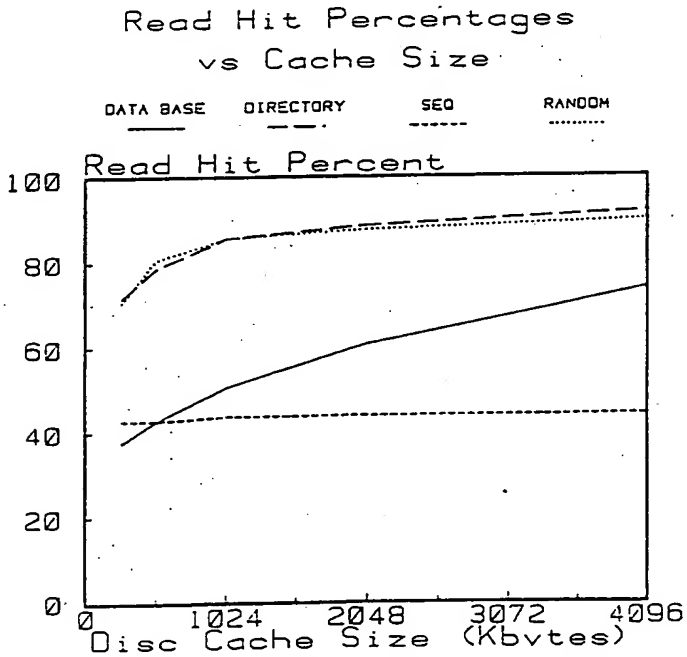

Fig. 13. *Potential benefit of disc caching for the HP 3000 family is shown by this graph of read hit rates that can be achieved by various access methods as a function of cache capacity*

modeling results shown on page 27. The graphs show transaction throughput as a function of processor speed with conventional storage management (Fig. 9a), external disc caching (Fig. 9b), and caching of discs in the primary memory via explicit global disc caching (Fig. 9c) and file mapping (Fig. 9d). Assumptions are the previously specified workload characteristics, four 25-ms discs, read hit rates of 85% on disc file accesses, write wait probabilities of 0, 1K-byte mean transfers without caching, and 4K-byte transfers with LRU replacement with caching. All of the secondary storage caching alternatives are able to provide effective processor utilization at processor speeds several times those that can be effectively used without caching with equivalent processor/disc configurations. Further, disc caching in the primary memory, either explicit or through file mapping, is able to provide effective processor utilization at speeds twice those that can be effectively used by external caching, assuming the same overheads and hit and wait rates. This is a result of the access time differential between main and external cache storage and the higher degree of parallelism in cache access with primary storage caching.

The overhead of internal caching becomes negligible as processor speeds increase, whereas the overhead in external caches stays fixed. However, with slower processors and a balanced configuration leading to high processor utilization without caching secondary storage, explicit internal caching degrades performance relative to no caching. The added overhead of locating the disc region in memory and moving the data to the target area does not pay off when the processor utilization is high. This effect was observed in the HP 3000 internal caching measurements. The low-end family members degraded in performance when explicit internal disc caching was enabled. External caching outperforms explicit internal caching in high utilization ranges as well, since the data transfer is performed in parallel with processor use, so the move overhead does not consume valuable processor time.

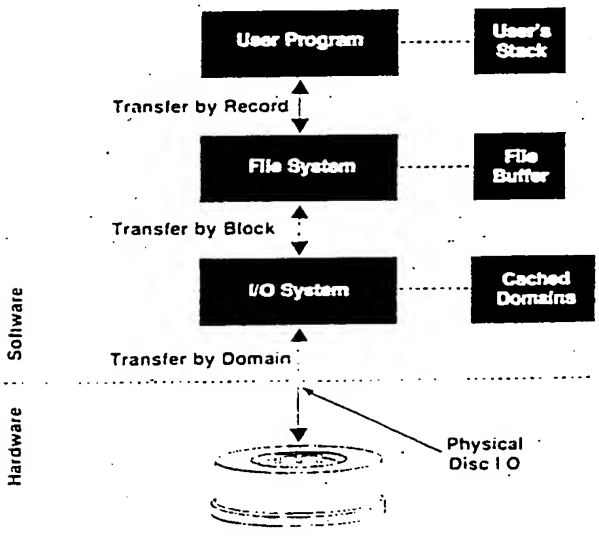

Fig. 14. *Explicit internal disc cache interfaces in the breadboard MPE operating system kernel developed for this study.*

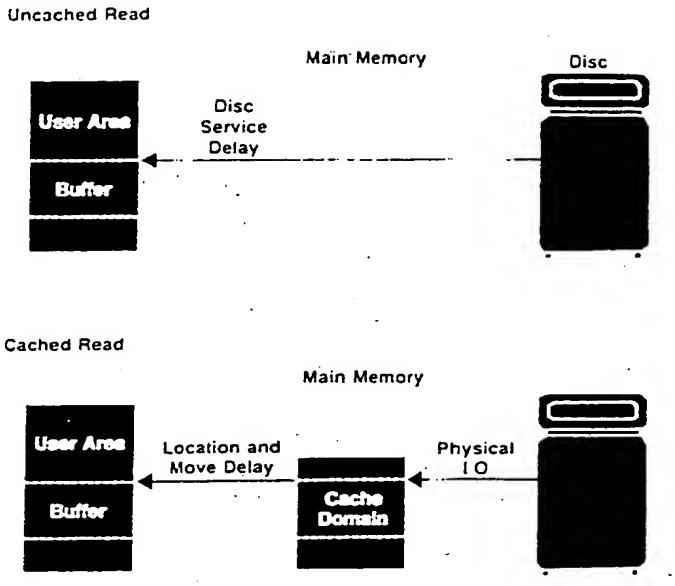

Fig. 15. *Disc read caching in the breadboard MPE kernel*

The dependency on disc access time and the number of discs is dramatically reduced with the secondary storage caching alternatives. This is shown for file mapping in Fig. 10 for the workload, disc subsystem, and read hit and write wait probabilities used in the previous modeling results. The behaviors of external caching and explicit internal caching are similar. By contrast with the linear dependency on the number of discs seen in Fig. 2 for a 3-MIPS processor without secondary storage caching, full processor utilization can be achieved with only a few discs at 7 MIPS with secondary storage caching. Reducing disc access time from 25 ms to 10 ms increases the effective utilization range from 1 MIPS to 4 MIPS without caching (Fig. 2b). With caching, reducing the mean disc service time extends the effective utilization beyond 30 MIPS with file mapping (Fig. 10b), but processors through 16 MIPS can be effectively used with a few 25-ms discs (Fig. 10a), provided that adequate read hit rates and write wait probabilities can be achieved.

The performance impact of read hit rates and write wait probabilities with systems employing secondary storage caching is significant. Fig. 11 shows the sensitivity of file mapping to the read hit and write wait probabilities. These are shown for two processor speeds, 3 MIPS and 7 MIPS. The sensitivity to read hit and write wait probabilities increases with processor speed, and there is a rapid drop in effective processor utilization for higher-speed processors as the read hit probability decreases and the write wait probability increases. This sensitivity to read hit rate and write wait probability decreases with reductions in disc access time. Thus faster drives enhance caching alternatives, and indeed are required if sufficiently high read hit rates and sufficiently low write wait probabilities cannot be achieved. The characteristic dependencies of the other caching alternatives on read hit rates and write wait probabilities are proportional to those presented here for file mapping.

Special mechanisms can be employed to achieve high read hit rates and low write wait rates. Since the secondary storage caching alternatives are sensitive to these parameters, it is important to exploit them.

Integrating cache fetch, replacement, and write handling with transaction management requirements is achievable with internal caching as described below. With external caching, on the other hand, this would be difficult to achieve, and high overhead would be incurred because of the protocols required between the host and the external device. Consequently, cache memory utilization and the sustained effective multiprogramming level of internal caching can be superior to that achievable with external caching.

With internal caching, the size of a fetched disc domain on a read miss can be tailored to the structure of the data, based on knowledge of the storage layout (e.g., fetch extents instead of fetching tracks containing unrelated data or only pieces of the required extent). The replacement policy can exploit operating system knowledge of access patterns (e.g., the policy can flush a cached disc domain from the cache memory after sequential reference and on file purging). Write posting order and write and read priorities can be adjusted to meet the current needs of transaction management.

Evaluation With Respect To Other Factors

Development of multiple local buffer management schemes is redundant, and the buffering capacity is localized and unresponsive to current memory loading conditions. The amount of memory devoted to the buffering of a specific file or subsystem would likely be either excessive or insufficient at any given moment, depending on the current memory availability and the current workload demand and priority structure. The problems are analogous to those encountered with older memory replacement policies based on fixed partitioning.

Global file buffers require a fixed partition of main memory between swap space and buffer space, and therefore are not responsive to dynamic memory load conditions. They suffer from fixed-partition problems as do local file buffers. Furthermore, they provide memory management functions, including space allocation, replacement, and disc access initiation and interrupt fielding. Therefore, developing efficient algorithms for the global buffer manager is redundant to the development of the manager of memory for transient code and data. Moreover, as is discussed in the next section, global file buffers are not amenable to supporting efficient local caching in decentralized systems.

Explicit internal disc caching and file mapping overcome the fixed-partition problems, so they can be expected to achieve higher read hit rates with dynamic workloads. Furthermore, it is easier to integrate disc access posting order and priority adjustments with these schemes.

Explicit internal disc caching in main memory requires a translation mechanism to locate a required cached disc domain in main memory. In large main memories there can be several thousand cached disc domains present at any moment, and locating a cached domain through a translation of disc address to main memory address can be expensive in processor cycles. Additionally, if the processor has a cache, the location sequence can flush a large portion of the cache while chasing through list structures. However, explicit internal disc caching can be performed with any processor architecture.

File mapping appears to offer the best of the alternatives for caching of discs in main memory. It leverages the main memory management mechanisms, has no limits on main memory space applied to file caching, and has no overhead for location of file domains in main memory. Applicability of this approach is limited, however, to processor architectures with large virtual address spaces ($\geq 2^{48}$ bytes to map many large files) and amenable protection mechanisms to restrict access to mapped-in files. Stonebraker[22] discusses problems with performing transactional management on mapped files with 32 bits of address space. His studies were based on the VAX family. He found that the overhead of mapping in and out pieces of files to accommodate the protection and space limitations exceeded the overhead of explicit data base buffering.

Evaluating with respect to cost, using faster discs and more of them to overcome the performance limitations without caching is clearly not cost-effective. External caching is more expensive than internal caching since additional power, cooling, cabinetry, and electronics are required in addition to the extra memory required for the

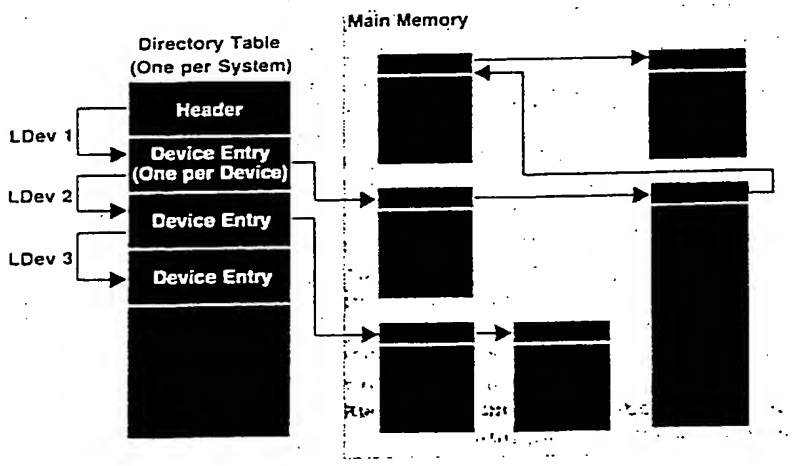

Fig. 16. *Cached disc domain location mechanism used in the breadboard MPE kernel. A separate list is maintained for each disc, identifying the memory regions corresponding to currently cached domains from that disc.*

caching. Also, more memory would be required to achieve the same hit rates in external caching unless the cache management is integrated with the operating system policies.

Evaluating with respect to reliability, internal caching introduces no new hardware components into the system. The reliability is identical to the uncached system, whereas any of the external cache architectures necessarily degrades system reliability by introducing additional hardware components. The software/firmware complexity of explicit internal and external caching is roughly the same, so reliability degradation because of cache management is comparable for these alternatives, while file mapping is simpler and more reliable. With external caching, the posting strategy of the peripheral cache is not integrated with the system posting strategy, so a consistent level of integrity is not guaranteed for transactional data base systems unless post order and wait for media update are observed, thereby impacting the write handling performance of peripheral cache alternatives.

Applicability to the HP 3000 Family

The research and development leading to the MPE-IV kernel[23] examined alternative algorithms for processor, main memory, disc, and semaphore management. This research focused on the interactions between algorithms managing these basic system resources, and determined an algorithm set for these resources that provides good performance through algorithm integration. The MPE-IV kernel is briefly described in the box on page 25. Above the management of these basic system resources, subsystems and applications manage data to provide extended system functionality. The data is kept in structured permanent and temporary objects including files, data bases, stacks, and heaps. Concurrent data access is managed through locking or versioning. Recovery from transaction aborts or system failures is handled through checkpointing, write-ahead logging, and roll-forward/roll-back recovery. To exploit the system price/performance potential offered by evolving

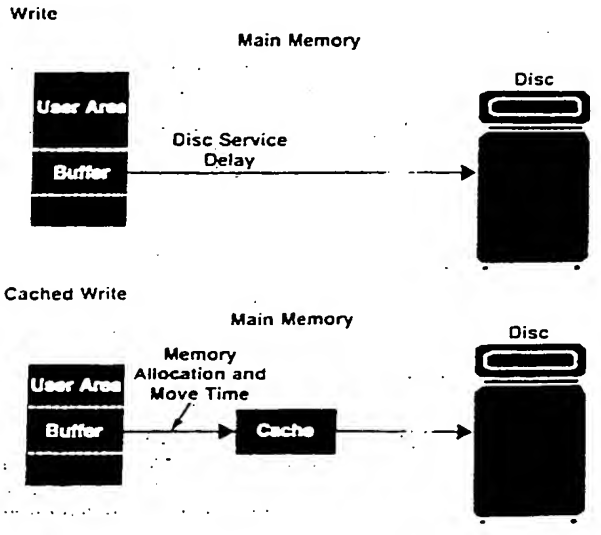

Fig. 17. *Disc write caching in the breadboard MPE kernel.*

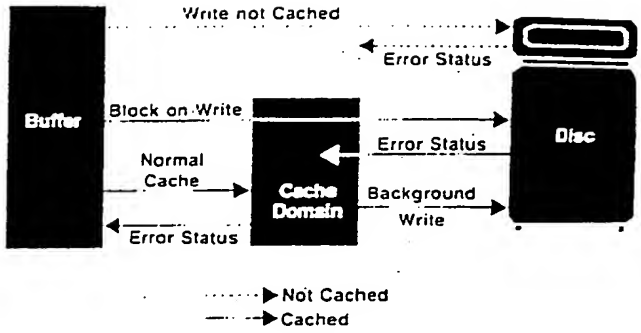

Fig. 18. *Write protocol for internal disc caching in the HP 3000*

technology, basic kernel main memory, disc, and processor management needs to be extended and integrated with subsystem and application data and recovery management. This was well demonstrated in our follow-on research to MPE-IV.

After the management strategies for main memory, disc, processor, and semaphore resources were integrated in MPE-IV to produce balanced resource utilization and significant performance improvement across the HP 3000 family, a new high-end system, the Series 64, was introduced. Its processor is twice as fast and its memory is four times as large as the previous top of the line. The system performance improvements realized with this computer were, however, sublinear with processor speed relative to the previous top-of-the line system. System performance with this high-end computer was found to be very sensitive to disc subsystem throughput and access times, but relatively insensitive to main memory capacity. In spite of the integrated strategies of the breadboard kernel, system performance was not scaling as the family was extended to significantly more powerful systems.

We investigated this scalability problem, and found it to be caused by a lack of scalability in the algorithms used for subsystem and application level buffering and recovery management. Processor utilization was limited by subsystems making processes wait for disc accesses to complete. The disc write accesses were being generated by local buffer replacement and write-ahead log management. The disc read accesses were initiated to resolve local buffer read misses.

We wished to extend the basic algorithm integration for processor, main memory, semaphore, and disc management to exploit current technology trends to realize significant improvements in system price/performance ratio. We examined disc buffering, intermediate storage levels between primary and secondary storage, disc caching in excess primary memory of the system processing unit, and integrating kernel resource management with higher-level data management.

Potential For Secondary Storage Caching

Our analysis, presented in the preceding sections, indicates that explicit internal caching is the best alternative for the HP 3000 family, given its architecture (object sizes must be ≤64K bytes). We proceeded to investigate whether

the workload characteristics were amenable.

Fig. 12 indicates the salient workload characteristics of the secondary storage referencing characteristics of the HP 3000 family. These characteristics were obtained using the measurement tools described in the box on page 23. Fig. 12a shows the transfer sizes for the major access methods (sequential, directory, data base). The distributions indicate that relatively small transfers are performed between primary and secondary storage by all the access methods, with 90% of the transfers less than 1K bytes. Fig. 12b shows the spatial locality of access method references to secondary storage. This stack depth distribution presents the probability of a reference at a given depth into a stack of disc references ordered on a least recently used basis. The distribution indicates that over 70% of all disc references are within the last 30 disc regions referenced. Fig. 12c shows the temporal locality of references of the access methods. It indicates that 50% of the disc references are to regions that have been referenced within the last second.

The resulting potential of secondary storage caching for the HP 3000 family is indicated in Fig. 13. This figure shows the read hit rates that can be achieved by the various access methods as a function of cache capacity. The data for this graph was obtained using the disc cache simulation model on disc access traces obtained from an internal HP production facility. For this installation, mean read hit rates of 85% are achievable with a few megabytes of memory applied to caching disc regions. Results using other trace data and field measurements of our system in use show read hit rates typically in the 65-85% range and read-to-write ratios ranging from 3:1 to 5:1.

The Explicit Internal Disc Cache Implementation

The MPE-IV kernel base was used to investigate principles and integrated approaches to caching discs in the main memory. Algorithm interactions were observed and improvements developed. Differences between disc and main memory caching and areas in which architectural improvements would be of benefit were noted.

The resource management mechanisms and strategies in the MPE-IV kernel provided an efficient, extensible research base. These strategies were extended to support explicit disc caching in primary memory. The resulting mechanisms and strategies integrate kernel and data management. Knowledge of file structure and access type is exploited to enhance prefetch and replacement decisions for disc domains. Data recovery protocols are supported without wait for posting through kernel adherence to post order constraints. Priorities of disc accesses are adjusted to reflect the changing urgency of read and write requests because of data management locks or commits. Process priorities are adjusted to reflect lock states.

The overall structure is shown in Fig. 14. The user program requests records. The file system maintains local file buffers, and on buffer misses or replacements, accesses the I/O system to initiate buffer reads or writes. Beneath the I/O interface, pieces of the disc memory are cached in main memory. Actual disc transfers are initiated by the cache manager.

Read Handling

Fig. 15 shows read processing with and without caching enabled. Without caching, the disc transfers data directly to the buffer, but a disc access delay is incurred. With caching, the disc transfers data to an area of memory reserved for the disc domain, and then the data is moved to the buffer using the processor. On read hits, data access requires locating the data in memory and moving it, rather than incurring a disc delay. This is performed on the current process's stack without a process switch.

Locating Cached Disc Regions

The cached disc domain location mechanism employed in the breadboard kernel is depicted in Fig. 16. A separate list is maintained for each disc that identifies the memory regions corresponding to currently cached domains from the disc. The list is ordered by increasing disc address, and a microcoded link-list-search (LLSH) instruction is used to locate a required region in the list.

This location scheme requires about 500 instructions for setup and cleanup, plus two memory references and two compares in the LLSH instruction for each cached domain in the list. Thus the overhead of translation increases with the memory size. This is not a particularly good feature. Thousands of domains can be cached for each disc in a large memory, so the overhead of translation can become significant. In hindsight, more attention should have been paid to the location mechanism. The overhead would be

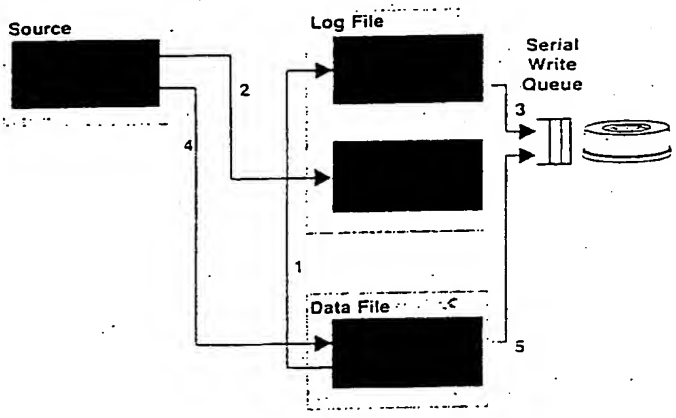

Fig. 19. *Sequence of operations for write-ahead logging with serial posting. This mechanism was developed to minimize the probability of waiting for disc writes to complete.*

reduced significantly by employing a hashing rather than a linked-list algorithm, but this is complicated by the variable size of cached disc regions. Architectures supporting file mapping in virtual space eliminate this overhead altogether.

Write Handling Mechanisms and Policies

Write handling with and without internal disc caching is shown in Fig. 17. Without caching, the transfer occurs directly to the disc from the buffer. The transfer can be initiated with or without wait, with explicit completion synchronization occurring later.

When a write is initiated with internal disc caching, either a disc domain already containing the specified disc addresses or a free area of memory is allocated and the data is moved into that area. A write to the disc is issued, and the process can continue to execute. The write is considered complete once the memory move occurs unless the file has been specified as requiring physical post before completion notification. Note that, thanks to the support of variable-sized segments in the HP 3000 architecture, a write does not require a fetch of the disc region. In paged systems, a write of a partial page requires that the page be fetched before the write can be initiated, thereby incurring an additional disc access on a write miss.

If there is currently a write pending against the specified disc domain, the process's request is queued until the pending write is posted to disc. The process may continue executing until it requests a wait for its write completion and the request is still queued. If the disc domain to be written is not currently cached, an available region of memory is obtained to map the corresponding disc image—i.e., no fetch of the disc domain to be written is required. When the move effecting the write takes place from the process's data area to the cached image of the disc, a post to the disc is initiated. Only the portion of the cached disc image that is modified by the write is posted. After the move to the disc image is performed and the post to disc is initiated, the writing process is allowed to continue running without having to wait for the physical post update to complete. This is all handled on the current process's stack, without even a process switch.

A write-through policy was chosen for our implementation of internal disc caching. The post request to disc is issued at a background disc priority. The priority of a pending post request is raised if the process waits for the post to complete or the region is required by the main memory replacement algorithm. Thus, issuing a physical post when the write is performed rather than waiting for replacement has no negative impact. Only idle disc capacity is used. Furthermore, pending writes of adjacent disc regions can be gathered into a single physical transfer, thereby reducing the total number of required physical accesses.

Write-through also has another benefit. The transaction recovery mechanisms require synchronization on physical commit at some point, so performing these early saves commit delays.

The write protocol is shown in Fig. 18.

We saw in the preceding sections the significant impact of having to wait for writes. To minimize the probability of waiting for disc writes to complete, we developed a special mechanism that allows the specification of posting order constraints at post time or on a file basis so that posting can proceed without wait if only a posting order constraint exists. With this facility, the post of the log can be issued on a nowait basis, and the data base write can be executed immediately. The kernel guarantees that posting order within a serial write queue matches the chronological order of post initiation. The sequence for write-ahead log use of this facility is depicted in Fig. 19.

To ensure disc consistency, only one write access for a serial write queue can be pending at a time. This is depicted in Fig. 20. Since this limits parallel use of available disc drives, multiple serial write queues for unrelated postings are beneficial.

Cache Fetch, Placement, and Replacement

The kernel's main memory placement and replacement mechanisms were extended to handle cached disc domains in the same manner as segments. Thus, cached disc domains can be of variable size, fetched in parallel with other segments or cached disc domains, garbage collected, and

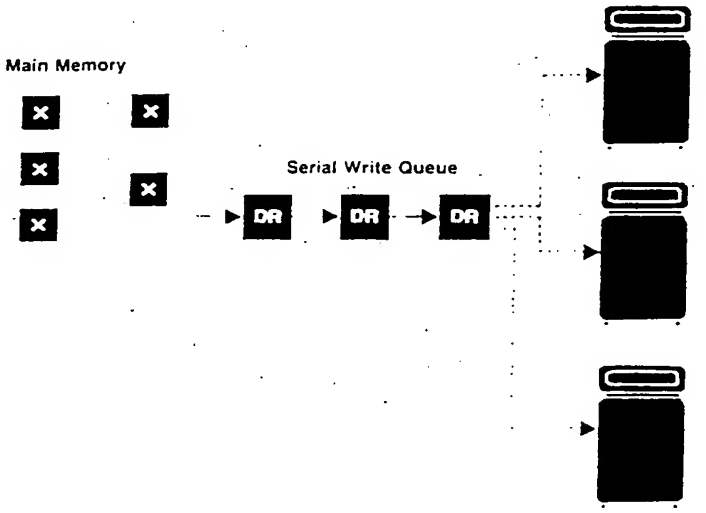

Fig. 20. *To ensure disc consistency, only one write access for a serial write queue can be pending at any time. Multiple serial write queues can be used to allow parallel use of available disc drives for unrelated postings.*

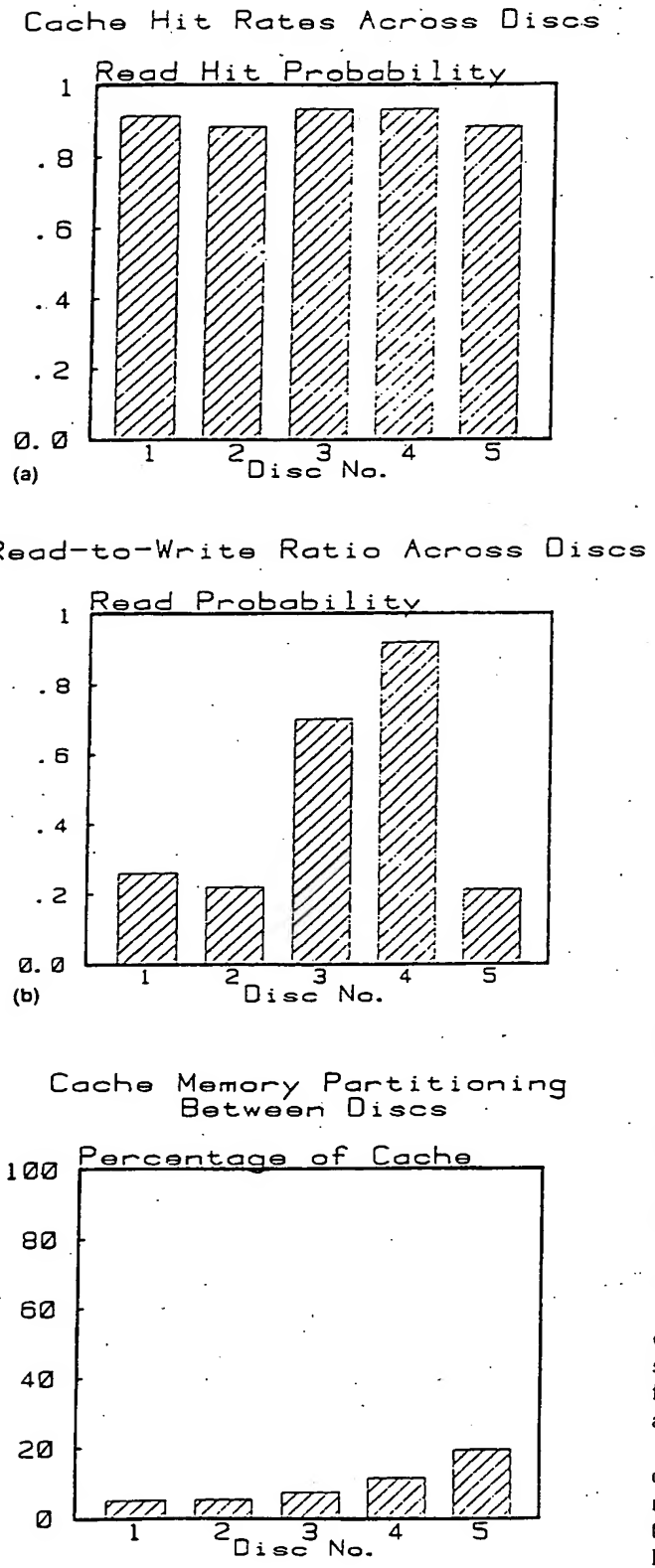

Fig. 21. *Dynamic partitioning of cache operations across the system discs of an HP 3000 system.*

replaced in an integrated manner with stacks, data segments, and code segments. The relative allocation of main memory between stack, data, code, and cached disc domain objects is entirely dynamic, responding to the current workload requirements and current memory availability.

Fetching and replacing differ from processor caching of main memories. Rao[24] discusses the impact of replacement algorithms on cache performance for processor caching of main memory, finding that the replacement algorithm has a secondary effect on cache performance.

With internal disc caching, when a request is made to read data that is not currently cached, the fetch strategy uses knowledge of file blocking, extent structure, access method, and current memory loading to select the optimal size of disc domain to be fetched into memory. The fetch is performed in an unblocked manner so that the requesting process or another process can run in parallel with the cache fetch from disc. With processor caches, the processor must idle on a cache miss since the process switch time exceeds the cache miss processing time.

No special treatment of cached domains is required for the replacement algorithm. It naturally protects the transient objects in smaller memories because of their smaller interreference times, and uses large memories for extensive write and read caching with the relative amount of cached disc space per device skewing naturally to the heavily referenced devices. This dynamic partitioning feature is very significant, since device skewing is common both in terms of referencing frequency and the spatial locality of references.

Fig. 21 shows the cache hit ratios, the read-to-write ratio, and the partitioning of main memory allocated for cached domains across the discs of an HP 3000 Computer System. The normal LRU-type replacement algorithm does fine in responding to the variable demand requirements of the various disc volumes.

Explicit prefetching and flushing for sequential access was found to improve hit rates through simulation studies on standard trace files, whereas special prefetching and flushing for other access modes did not.

This policy was implemented in the kernel. When a process finishes referencing a cached disc domain in sequential mode, the domain is flushed immediately from main memory since it won't be needed again. In this way, memory utilization is improved over that achievable with the kernel's standard LRU-type replacement algorithm.

External Caching Controls

The external controls for caching allow caching to be enabled/disabled against specific discs, display the current status of disc caching, set posting policies on a system and file basis, and control the roundoff fetch sizes for random and sequential access.

Defaults for the tuning parameters were selected based on simulations of disc access traces using the simulation model. Good defaults for random fetch sizes were found to be 4K bytes and for sequential fetch sizes, 24K bytes. Large prefetches were found to pay off significantly for sequential, but not for random type accesses. Rounding the fetching above the requested block was found to be superior for all access methods to fetching below the requested block

or centering on the requested block. The choice of tuning parameters is, as always, an adjustment to the access patterns of the particular subsystems and data bases.

Performance of the Disc Cache Implementation

With these mechanisms and strategies, the extended kernel significantly reduces the traffic between the main memory and secondary disc storage and significantly reduces delays in reading or writing disc information. Read hit rates up to 85% are common for file, data base, and directory buffer fills. These read hits eliminate up to 65% of the disc accesses with a 5:1 read-to-write ratio. Because of the caching of writes, most delays for posting are eliminated. Together, the read hits and cached writes eliminate 90% of process delays caused by disc accessing. This dramatically reduces semaphore holding times, which especially benefits data base systems.

The impact on system performance over the noncached kernel is shown in Fig. 22. Throughput improvements of 50% and response time reductions of 5:1 are standard on

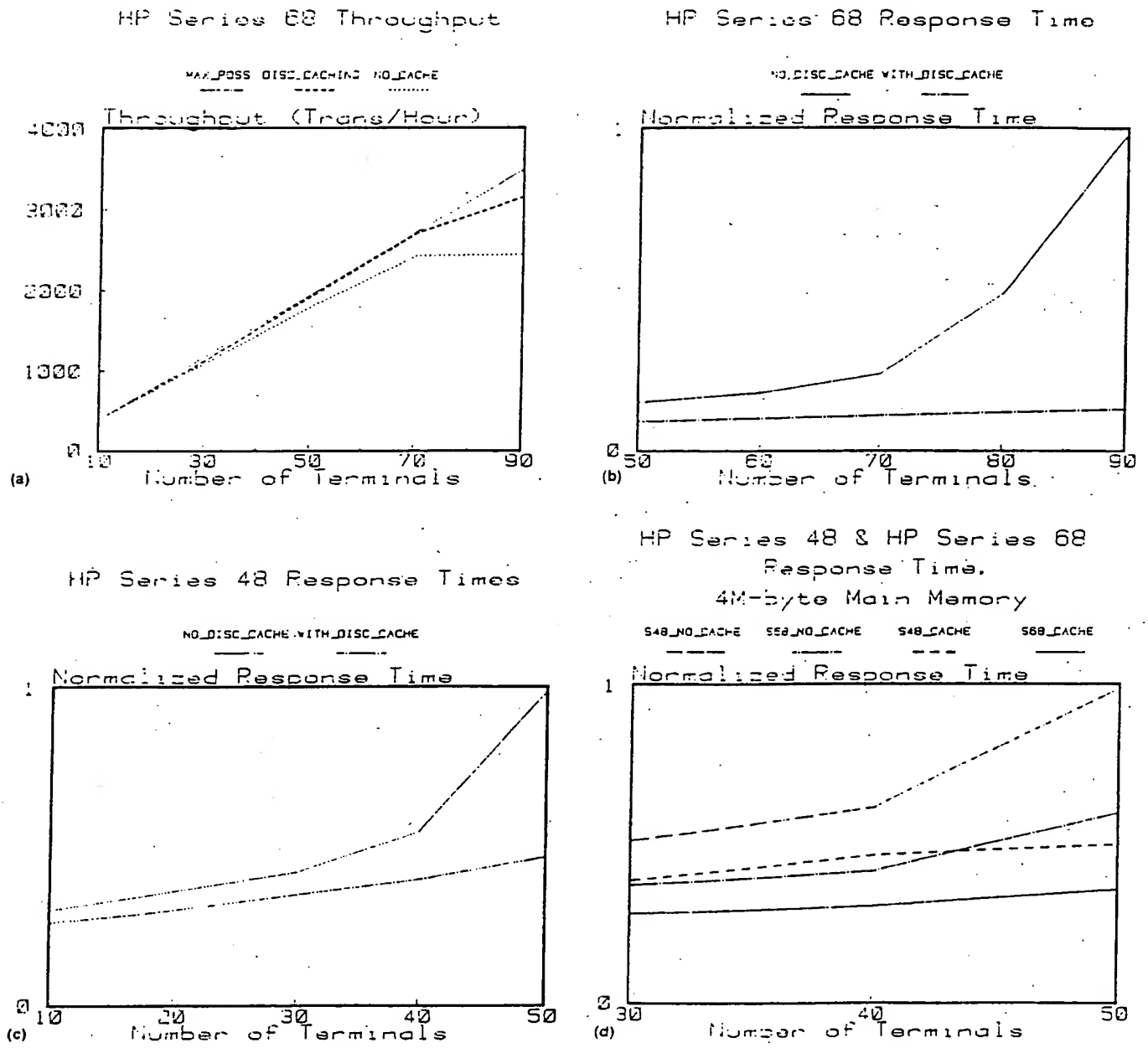

Fig. 22. *Performance of HP 3000 Series 48 and 68 systems with and without disc caching. Throughput improves most on the larger Series 68 system with caching.*

the high end (Series 68) while the midrange system (Series 48) gets about 25%. For the low-end system (Series III or 33, not shown in Fig. 22), performance actually degrades, thus demonstrating the scaling of performance with processor speeds. However, the midrange system with kernel disc caching outperforms the high-end machine without kernel disc caching.

Fig. 23 compares the kernel with and without caching for configurations with four 125M-byte discs or two 400M-byte discs. The cached system with two disc servers outperforms the uncached system with four disc servers. Moreover, the cached system performance is the same with two or four discs, thereby demonstrating that caching can exploit the system cost advantages of large-capacity discs.

A benchmark comparison[25] was made between an IBM 3033 (5 MIPS) and an HP 3000 Series 68 (1 MIPS) with and without applying disc caching in main storage. The benchmark ran Compserv's Materials Request and Planning software developed for both the HP 3000 and the IBM 3033 systems. The benchmark consists of a materials request and planning batch job that performs a large data base implosion to determine the parts, lead times, and orders to manufacture a specified set of products subject to a master production schedule. This type of application tends to be highly I/O intensive since it accesses the data base in an inverted manner, starting with schedules and searching for qualifying parts.

The benchmark ran in 28 hours on the IBM 3033, 49 hours on the HP 3000 Series 68 without kernel disc caching, and in 27.4 hours on the HP 3000 Series 68 with kernel disc caching. That the HP 3000 Series 68 could outperform the IBM 3033 in spite of the fivefold difference in processor speed indicates that caching by the kernel of disc domains in excess main storage with the cache management policies introduced in this research has potential application in other computer families employing locally managed, limited caching of data items by data base or file systems. Data base and file system caching is limited to a fixed capacity and applies policies that optimize for the standard access approach. When more resources are available, such as main memory for a stand-alone batch job, and access is nonstandard, as in this inverted-access case, localized caching policies do not respond.

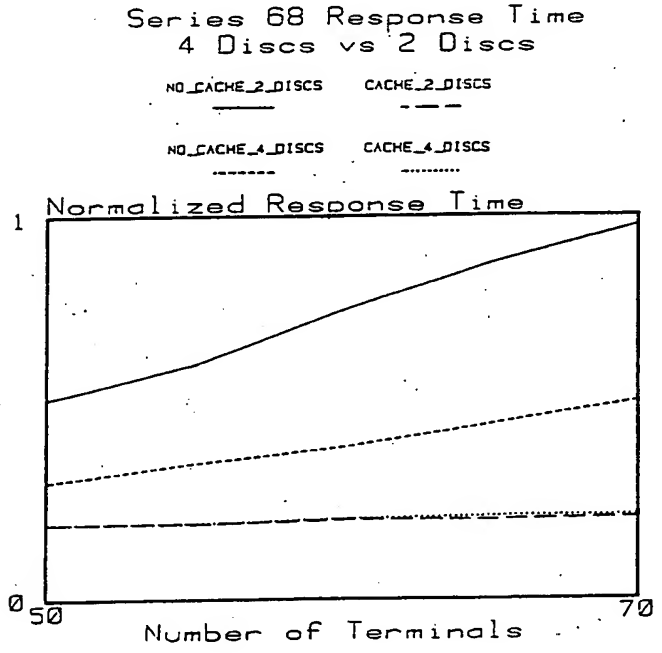

Fig. 23. *Effects of multiple discs and disc caching in the HP 3000 Series 68. Sensitivity to the number of discs is dramatically reduced by caching.*

In small memory systems under memory pressure and with slow processors, disc caching degrades performance. The cost of locating the cached disc domain in main memory and moving the domain gets added to the disc access time. With low hit rates and slow processors, this overhead exceeds the benefits of caching the discs. This overhead is not present in architectures supporting file mapping.

Conclusions

This research and development effort examined alternative approaches to exploit current trends in processor and memory technology to realize significant improvements in system price/performance ratio. The results were applied to the HP 3000 family.

Caching secondary storage in primary main memory through explicit internal caching or file mapping, coupled with integrated kernel and data management algorithms, was found to provide the most cost-effective and best performing alternative, given current technology trends. These approaches are supplemented by improvements in secondary storage device characteristics.

Secondary storage cache management can be integrated with data management to obtain significant incremental performance improvements. Specific improvements were identified by integrating write-ahead logging with disc access scheduling through the adherence to posting order constraints, bumping the priority of affected disc post requests at commit time, and using extent boundary and access type knowledge for selecting the fetch size on read misses and flushing on sequential access.

Effective utilization of high-speed processors requires sustaining a high multiprogramming level. Improved concurrency control schemes, such as late binding, granular locking for updates, and versioning for queries, help achieve this. Data base locking can be integrated with kernel resource management by raising the priority of a lock holder when a more urgent process queues on its lock.

Providing caching of discs in main memory differs from caching main memory for processors in several ways. The delay in resolving a read miss is highly state-dependent, requiring disc queueing and head positioning components. Since the read miss resolution time is long compared with the time required to state save the current process and launch another process, other ready processes can be run if the effective multiprogramming level is sufficiently high, allowing productive processor utilization while resolving read misses. Special knowledge of access patterns can readily be exploited to improve cache fetch and replacement strategies. Write misses can be resolved without requiring a fetch from the backup store if variable-sized allocation

is used, thereby using large caches to eliminate process waits on write misses. Special constraints exist on the write handling policies for updating the backup memory because of the requirements of write-ahead logging and shadow paging in transactional systems. Classical techniques used to model processor caches did well in analyzing hit rates, but more complex models were required to capture queueing and nowait effects found in secondary store caching.

The implementation of explicit internal disc caching for the HP 3000 family graphically demonstrates the results the analysis predicts. The effective utilization of higher-speed processors and reduced dependency on the number of discs is clearly demonstrated. The implementation demonstrates further that integrated internal caching of secondary storage can be accomplished with conventional hardware and software architectures. Disc data management integration with internal cache management can be achieved in existing systems by providing special functions to allow subsystems and applications to influence the management of caching, disc scheduling, and main memory management to meet their performance and recovery objectives.

We feel that the major contributions of this research and development effort are the performance analysis techniques, the characterization of secondary storage caching alternatives, and the mechanisms we developed to integrate the requirements of high-level data management with the basic kernel management of processor, disc, primary memory, and semaphore resources. Our leveraging of kernel mechanisms for transient code and data management resulted in reduced development time, highly parallel cache management, and a smooth, dynamic partition of main memory. Our extensive testing (over a year) in varied internal production sites resulted in a high-quality product. However, our implementation of the location mechanism is inefficient for large memories, and further work could be done on data bases and applications to exploit more fully the underlying caching.

The performance advantages of secondary storage caching will be needed in decentralized as well as centralized transaction systems. Caching secondary storage in decentralized systems introduces problems of cache consistency analogous to those encountered in multiprocessors with private caches. Analyzing and building solutions to realize the performance benefits of caching secondary storage in decentralized configurations presents us with a challenging follow-on to the applied research reported here.

Acknowledgments

We'd like to acknowledge our R&D manager Howard Smith and our marketing manager Nancy Anderson for recognizing the potential of our project and supporting its delivery. Our thanks go to Jeff Byrne, our product manager, and to Sam Boles and Becky McBride of the Business Development Group, who were really helpful in providing ongoing encouragement and support.

References

1. J.R. Busch and A.J. Kondoff, "MPE Disc Cache: In Perspective," *Proceedings of the HP 3000 International User's Group Conference*, Edinburgh, October 1983, reprinted in *Journal of the HP 3000 International Users Group*, Vol. 7, no. 1, January-March 1984, pp 21-24.

2. A.E. Bell, "Critical Issues in High-Density Magnetic and Optical Data Storage," *Laser Focus/Electrooptics*, August 1983, pp 61-66 and September 1983, pp 125-136.

3. R. Roseberg, "Magnetic Mass Storage Densities Rise," *Electronics*, October 29, 1984.

4. M.J. Flynn, J.N. Gray, A.K. Jones, K. Legally, H. Opderbeck, G.J. Popek, B. Randell, J.H. Saltzer, and H.R. Wiehle, "Operating Systems : An Advanced Course," *Lecture Notes In Computer Science*, Springer-Verlag, 1978.

5. W.W. Chu and P.P. Chen, *Tutorial : Centralized and Distributed Data Base Systems*, IEEE Computer Society.

6. H.T. Kung and J.T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems*, Vol. 6, no. 2, June 1981, pp 213-226.

7. R. Agrawal, M.J. Carey, and D.J. DeWitt, *Deadlock Detection Is Cheap*, Draft, Computer Science Department, University of California, Berkeley, 1983.

8. A. Chan and R. Gray, "Implementing Distributed Read Only Transactions," submitted for publication, 1983.

9. A.J. Smith, "A Modified Working Set Paging Algorithm," *IEEE Transactions on Computers*, Vol. C-25, no.9, September 1976, pp 907-913.

10. "Series/1 Gets Top CPU, Extended Networking; System/38 Also Capped," *Computerworld*, Vol. 17, no. 15, April 1983.

11. U. Krastins, "Cache Memories Quicken Access to Disk Data," *Electronic Design*, May 1982, pp 41-44.

12. H.R. Crouch and J.B. Cornett, Jr., "CCDs in Memory Systems Move into Sight," *Computer Design*, September 1976, pp 75-80.

13. G. Panigrahi, "Charge-Coupled Memories for Computer Systems," *Computer*, April 1976, pp 33-42.

14. H. Chang, "Capabilities of the Bubble Technology," *Proceedings of the National Computer Conference*, 1974, pp 847-855.

15. T.C. Chen, "Magnetic Bubble Memory and Logic," *Advances in Computers*, Vol. 17, 1978, pp 224-282.

16. W. Hugelshofer and B. Schultz, "Cache Buffer for Disk Accelerates Minicomputer Performance," *Electronics*, February 1982, pp 155-159.

17. M. K. Mc Kusick, S.J. Leffler, and W.N. Joy, *The Implementation of a Fast File System for UNIX*, Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

18. "Cache/Q by Techne the Software Accelerator," Techne Software Corporation, advertisement, 1983.

19. E.E. Organik, *The Multics System*, MIT Press, 1972.

20. E. Basart, D. Folger, and B. Shellooe, "Pipelining and New OS Boost Mini to 8 MIPS," *Mini-Micro Systems*, September 1982, pp 258-269.

21. J.N. Gray, "Notes on Data Base Operating Systems," *Operating Systems: An Advanced Course*, edited by R. Bayer, M. Graham, and G. Seegmuller, Springer-Verlag, 1979, pp 393-481.

22. M. Stonebraker, "Virtual Memory Transaction Management," Memorandum No. UCB/ERL M83/74, Electronics Research Laboratory, University of California, Berkeley, December 19, 1983.

23. J.R. Busch, *Integrated Resource Management Algorithms for Computer Systems*, Ph.D. Dissertation, University of California at Los Angeles, 1984. University Microfilms International, Ann Arbor, Michigan, no. 8420156.

24. G.S. Rao, "Performance Analysis of Cache Memories," *Journal of the Association for Computing Machinery*, Vol. 25, no. 3, July 1978, pp 378-395.

25. "Series 64 With Disc Caching Beats IBM 3033 in Batch MRP Run," *Hewlett-Packard Computer News* (internal publication), October 1, 1983.